Secure VM Service Module for SEV-SNP Guests

Guest Communication Interface

Publication # **58019** Revision: **0.50**	
Issue Date: **August 2022**	

Advanced Micro Devices

Specification Agreement

This Specification Agreement (this "Agreement") is a legal agreement between Advanced Micro Devices, Inc. ("AMD") and "You" as the recipient of the attached AMD Specification (the "Specification"). If you are accessing the Specification as part of your performance of work for another party, you acknowledge that you have authority to bind such party to the terms and conditions of this Agreement. If you accessed the Specification by any means or otherwise use or provide Feedback (defined below) on the Specification, You agree to the terms and conditions set forth in this Agreement. If You do not agree to the terms and conditions set forth in this Agreement, you are not licensed to use the Specification; do not use, access or provide Feedback about the Specification.

In consideration of Your use or access of the Specification (in whole or in part), the receipt and sufficiency of which are acknowledged, You agree as follows:

1. You may review the Specification only (a) as a reference to assist You in planning and designing Your product, service or technology ("Product") to interface with an AMD product in compliance with the requirements as set forth in the Specification and (b) to provide Feedback about the information disclosed in the Specification to AMD.
2. Except as expressly set forth in Paragraph 1, all rights in and to the Specification are retained by AMD. This Agreement does not give You any rights under any AMD patents, copyrights, trademarks or other intellectual property rights. You may not (i) duplicate any part of the Specification; (ii) remove this Agreement or any notices from the Specification, or (iii) give any part of the Specification, or assign or otherwise provide Your rights under this Agreement, to anyone else.
3. The Specification may contain preliminary information, errors, or inaccuracies, or may not include certain necessary information. Additionally, AMD reserves the right to discontinue or make changes to the Specification and its products at any time without notice. The Specification is provided entirely "AS IS." AMD MAKES NO WARRANTY OF ANY KIND AND DISCLAIMS ALL EXPRESS, IMPLIED AND STATUTORY WARRANTIES, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, TITLE OR THOSE WARRANTIES ARISING AS A COURSE OF DEALING OR CUSTOM OF TRADE. AMD SHALL NOT BE LIABLE FOR DIRECT, INDIRECT, CONSEQUENTIAL, SPECIAL, INCIDENTAL, PUNITIVE OR EXEMPLARY DAMAGES OF ANY KIND (INCLUDING LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, LOST PROFITS, LOSS OF CAPITAL, LOSS OF GOODWILL) REGARDLESS OF THE FORM OF ACTION WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE) AND STRICT PRODUCT LIABILITY OR OTHERWISE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
4. Furthermore, AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur.

5. You have no obligation to give AMD any suggestions, comments or feedback (“Feedback”) relating to the Specification. However, any Feedback You voluntarily provide may be used by AMD without restriction, fee or obligation of confidentiality. Accordingly, if You do give AMD Feedback on any version of the Specification, You agree AMD may freely use, reproduce, license, distribute, and otherwise commercialize Your Feedback in any product, as well as has the right to sublicense third parties to do the same. Further, You will not give AMD any Feedback that You may have reason to believe is (i) subject to any patent, copyright or other intellectual property claim or right of any third party; or (ii) subject to license terms which seek to require any product or intellectual property incorporating or derived from Feedback or any Product or other AMD intellectual property to be licensed to or otherwise provided to any third party.
6. You shall adhere to all applicable U.S., European, and other export laws, including but not limited to the U.S. Export Administration Regulations (“EAR”), (15 C.F.R. Sections 730 through 774), and E.U. Council Regulation (EC) No 428/2009 of 5 May 2009. Further, pursuant to Section 740.6 of the EAR, You hereby certifies that, except pursuant to a license granted by the United States Department of Commerce Bureau of Industry and Security or as otherwise permitted pursuant to a License Exception under the U.S. Export Administration Regulations ("EAR"), You will not (1) export, re-export or release to a national of a country in Country Groups D:1, E:1 or E:2 any restricted technology, software, or source code You receive hereunder, or (2) export to Country Groups D:1, E:1 or E:2 the direct product of such technology or software, if such foreign produced direct product is subject to national security controls as identified on the Commerce Control List (currently found in Supplement 1 to Part 774 of EAR). For the most current Country Group listings, or for additional information about the EAR or Your obligations under those regulations, please refer to the U.S. Bureau of Industry and Security’s website at *http://www.bis.doc.gov/*.
7. If You are a part of the U.S. Government, then the Specification is provided with “RESTRICTED RIGHTS” as set forth in subparagraphs (c) (1) and (2) of the Commercial Computer Software-Restricted Rights clause at FAR 52.227-14 or subparagraph (c) (1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7013, as applicable.
8. This Agreement is governed by the laws of the State of California without regard to its choice of law principles. Any dispute involving it must be brought in a court having jurisdiction of such dispute in Santa Clara County, California, and You waive any defenses and rights allowing the dispute to be litigated elsewhere. If any part of this agreement is unenforceable, it will be considered modified to the extent necessary to make it enforceable, and the remainder shall continue in effect. The failure of AMD to enforce any rights granted hereunder or to take action against You in the event of any breach hereunder shall not be deemed a waiver by AMD as to subsequent enforcement of rights or subsequent actions in the event of future breaches. This Agreement is the entire agreement between You and AMD concerning the Specification; it may be changed only by a written document signed by both You and an authorized representative of AMD.

Acknowledgements

AMD would like to acknowledge Jon Lange for his work in creating the initial draft of the SVSM specification. Thank you, Jon.

Contents

List of Tables

Revision History

Date	Revision	Description
August 2022	0.50	• Initial public release

1 Abstract

AMD's Secure Encrypted Virtualization with Secure Nested Paging (SEV-SNP) offers powerful and flexible support for isolation of a guest Virtual Machine (VM) state from an untrusted host operating system. The VM Permission Level feature permits the inclusion of components in the guest that can run with a higher privilege than the guest operating system, offering an environment for secure, privileged code modules to run without interference from the bulk of the guest. Such code modules are not part of the guest operating system and may be designed to be modular, offering compatibility with a wide variety of guest configurations. For such a modular design to be successful, a standard calling convention must exist to permit the guest operating system and secure modules to communicate without interference from the untrusted host environment. This document defines the standard by which these modules can exist together and communicate within a single SNP guest.

2 Scope of Document

This document defines a Secure VM Service Module (SVSM) as an environment that can host privileged modules within the guest, and it also defines mechanisms by which a guest operating system can determine the presence of the SVSM, the set of services offered by the SVSM, and the mechanism by which the guest can communicate with those services.

This document does not describe the internals of the SVSM. Multiple SVSM implementations are encouraged; this document defines standards by which an SVSM communicates with a guest operating system so that multiple SVSM implementations can be compatible with multiple guest operating systems.

This document does not describe the mechanism by which the Virtual Machine Manager (VMM) loads the SVSM. That mechanism is assumed to be specific to the host architecture. The SVSM is expected to be loaded and measured as part of the initial guest image so that the specific identity of the SVSM associated with a guest can be verified through measurement and attestation.

This document does not describe mechanisms by which the VMM communicates with the SVSM nor chooses to invoke the SVSM. Those details are assumed to be specific to the host environment, though they could be standardized under a different specification.

This document does not describe the threading model of the SVSM. Some SVSM implementations may choose a separate execution context (a unique VMSA) per guest vCPU, while other SVSM implementations may choose a single execution context that services all guest vCPUs. The negotiation of threading models between the SVSM and the host as well as the mechanism by which the host indicates which guest vCPU makes a request are assumed to be specific to the host environment, though they could be standardized under a different specification.

3 Environment

The SVSM is expected to execute in VMPL0 of the guest. To ensure privilege separation for security-sensitive services, the bulk of the guest is expected to run at a VMPL other than zero. The SVSM must offer services to proxy requests that would normally be made by a guest running at VMPL0 but which are architecturally impossible when the guest is running at a VMPL other than zero (e.g., use of the PVALIDATE instruction and certain forms of RMPADJUST). Those services form part of the "core protocol." (See the "Core Protocol" section on page 17.)

The SVSM image and its initial data are expected to occupy a contiguous portion of the gPA space of the guest. That memory must be configured with VMPL permissions that grant access to VMPL0 but no lower VMPL. The initial SVSM memory configuration must be sufficient to offer all required SVSM services. A mechanism is defined by which the SVSM can obtain additional memory if required to support additional services that may be requested by the guest OS.

4 Discovery

4.1 Boot

When the guest first starts, it is expected to execute in the context of the SVSM before any lower VMPL is started. This will permit the SVSM to initialize itself and make itself discoverable.

The SVSM advertises its presence by writing information into the secrets page, as described in the following table. Note that the portions of the secrets page at byte offsets described here are always zeroed during initial construction of the secrets page and reserved for use by the SVSM. They will always be zero for every guest unless they are initialized by the SVSM.

Table 1: Secrets Page Fields

Byte Offset	Size	Field Name	Description
0x140	8 bytes	SVSM_BASE	Base gPA of the SVSM area. This must be a multiple of 4 KB.
0x148	8 bytes	SVSM_SIZE	Number of bytes in the SVSM area. This must be a multiple of 4 KB.
0x150	8 bytes	SVSM_CAA	gPA of an 8-byte area used for guest/SVSM communication.
0x158	4 bytes	SVSM_MAX_VERSION	Maximum version of the core protocol supported by the SVSM.
0x15C	1 byte	SVSM_GUEST_VMPL	Indicates the VMPL at which the guest is executing.
0x15D	3 bytes		Reserved.

Note that the SVSM is expected to zero the portion of the secrets page that contains the VMPCK associated with VMPL0, to prevent the guest OS from intruding on any conversation between the SVSM and the PSP.

When the guest OS starts, it must first read the SVSM_SIZE field. If this field is zero, then no SVSM is present. If this field is non-zero, indicating that an SVSM is present, then the guest must note the memory range spanned by the SVSM so that it does not attempt to use any memory in that range. (It may, for example, choose to identify that memory as EfiPlatformReserved.) It must capture the SVSM_CAA value for use in communication with the SVSM.

This specification assumes that the initial guest image contains only a single VMSA, used for the startup vCPU. If the guest requires additional VMSAs, they should be created dynamically. (See the section “SVSM_CORE_CREATE_VCPU Call” on page 18.)

4.2 Post-Boot

After boot, it may still be necessary for guest components that do not have access to the secrets page (e.g., UEFI runtime services invoked after the guest OS has started and taken over the SVSM interface) to perform calls to the SVSM. For those components, an alternate discovery mechanism is required.

To support separate components, a discovery mechanism is defined that requires specific handling of #VC exceptions. It is assumed that if the SVSM was discovered at boot, then the guest component that implements the #VC handler is also aware of the SVSM and the location of its Calling Area.

To discover whether an SVSM is present, the separate component must execute CPUID(EAX=8000_001F). This will result in a #VC exception, and the #VC handler is expected to set EAX[28]=1 in the response. When the component that executed the CPUID observes EAX[28]=1 in the response, it will know that an SVSM is reachable. This CPUID response bit is reserved for the purpose of enumerating the presence of an SVSM and will never be set by hardware or in any CPUID data generated by the PSP.

To discover the location of the SVSM Calling Area, the separate component must read MSR C001_F000. This will result in a #VC exception, and the #VC handler is expected to supply the gPA of the SVSM Calling Area as the MSR value. Writes to the MSR are not expected, and the #VC handler is not required to handle writes to the MSR. This MSR is reserved for the purpose of exposing the gPA of the SVSM Calling Area and will never be implemented in hardware.

Once the SVSM Calling Area has been located, the separate component can issue calls to the SVSM normally. The separate component should not alter the state of the SVSM in a way that is not expected by the remainder of the guest (e.g., no component should issue the SVSM_CORE_REMAP_CA command).

5 Calling Convention

Each call to the SVSM conveys data through a combination of the SVSM Calling Area (whose address was first configured through the SVSM_CAA field of the secrets page) and registers. Use of the Calling Area is necessary to ensure that the SVSM can tell the difference between a call that was issued by the guest and a spurious invocation by a poorly behaved host. Registers are used for all other parameters.

The initially configured SVSM Calling Area is a page of memory that lies outside the initial SVSM memory range and has not had its VMPL permissions restricted in any way. The address is guaranteed to be aligned to a 4 KB boundary, so the remainder of the page may be used by the guest for memory-based parameter passing if desired.

The contents of the Calling Area are described in the following table:

Table 2: Calling Area

Byte Offset	Size	Name	Description
0x000	1 byte	SVSM_CALL_PENDING	Indicates whether a call has been requested by the guest (0=no call requested, 1=call requested).
0x001	1 byte	SVSM_MEM_AVAILABLE	Free memory is available to be withdrawn.
0x002	6 bytes		Reserved. The SVSM is not required to verify that these bytes are zero.

Each call is identified by a 32-bit protocol number and a 32-bit call identifier specific to the protocol. The Core Protocol is identified by protocol number zero, and additional protocols have uniquely assigned values.

To make a call to the SVSM, the guest OS must load the RAX register with the identifier of the call, where bits [63:32] hold the protocol number and bits [31:0] hold the call identifier within the protocol. Additional registers and/or memory may need to be configured with values specific to the call being issued. Once all memory and registers have been prepared, the guest OS must write a value of 1 to the SVSM_CALL_PENDING field of the Calling Area to indicate its readiness to issue the call. Finally, the guest OS must execute VMGEXIT to request that the host execute the SVSM on behalf of the calling vCPU. This request is made in one of two ways, either by using the GHCB MSR protocol with a value of 0x16 or by setting GHCB fields SW_EXITCODE=0x8000_0017 and SW_EXITINFO1=0. (See the GHCB Specification for further details.)

When the SVSM receives the call, it is expected to set VMSA.EFER.SVME=0 for the calling vCPU of the guest OS; this ensures that the host cannot attempt to reenter the calling vCPU while SVSM call processing is underway. (An attempt to enter the guest would result in a failure due to invalid VMSA contents.) The SVSM is then expected to examine the SVSM_CALL_PENDING field to determine whether any call was actually requested by the guest OS; if the host illegally entered the SVSM, this field will be zero. In such a case, no action will be taken by the SVSM other than setting VMSA.EFER.SVME=1 for the calling vCPU and returning to the guest. In addition, the SVSM is expected to examine VMSA.EXITCODE after setting VMSA.EFER.SVME=0 to ensure that the guest is on the expected VMGEXIT instruction boundary before proceeding. If the exit code does not indicate exiting due to VMGEXIT, the SVSM should reset VMSA.EFER.SVME=1 and take no further action before returning to the guest.

Once the SVSM determines that a calling request is legitimate, it will read the value of RAX from the VMSA of the requesting vCPU and process the call accordingly. Upon completion of the call, the SVSM will set RAX in the VMSA of the requesting vCPU to indicate the result of the call. It will clear SVSM_CALL_PENDING in the Calling Area to indicate that the call was completed, set VMSA.EFER.SVME=1 for the calling vCPU (only after VMSA.RAX and SVSM_CALL_PENDING fields have been set), and return to the guest.

Upon its return, the guest must atomically clear the SVSM_CALL_PENDING field and examine the previous value. The guest cannot trust that the host has executed the SVSM call as desired, nor can it assume that the host will not attempt to execute the SVSM call at an inopportune time, so the guest must clear the pending request at the same time that it extracts the previous value for examination. If the previous value of SVSM_CALL_PENDING was non-zero, the guest knows that the SVSM never executed the call and must either retry the call or accept the fact that the host did not honor the request to execute the SVSM. If the previous value of SVSM_CALL_PENDING was zero, then the guest knows that the call completed and can examine the value of RAX to determine whether the call completed successfully.

Result values returned in RAX are 32-bit values (a 64-bit sign extension is ignored) divided into three categories: successful completion with distinct completion information, unsuccessful completion for a specified reason, and requests for additional memory. Most result codes are specific to individual protocols, but a portion of the result space is reserved for common values.

Table 3: Result Codes

Result code	Name	Meaning
0x0000_0000	SVSM_SUCCESS	The call completed successfully.
0x0000_0000 - 0x0000_0FFF		Reserved for future use.
0x0000_1000 - 0x3FFF_FFFF		Defined by the protocol that was invoked.

Result code	Name	Meaning
0x4000_0000 - 0x7FFF_FFFF		Additional memory is required to complete the requeste operation. Bits 29:0 indicate the number of 4 KB pages th are required.
0x8000_0000	SVSM_ERR_INCOMPLETE	The requested call was partially performed. The guest mu request additional processing by setting SVSM_CALL_PENDING and invoking the SVSM again.
0x8000_0001	SVSM_ERR_UNSUPPORTED_PROTOCOL	The requested protocol is not supported.
0x8000_0002	SVSM_ERR_UNSUPPORTED_CALL	The requested call ID is not supported by the requested protocol.
0x8000_0003	SVSM_ERR_INVALID_ADDRESS	A gPA specified as part of a call is invalid.
0x8000_0004	SVSM_ERR_INVALID_FORMAT	A reserved value was specified in SVSM_CALL_PENDING.
0x8000_0005	SVSM_ERR_INVALID_PARAMETER	One or more invalid parameters were specified to a call.
0x8000_0006	SVSM_ERR_INVALID_REQUEST	The request cannot be supported by the protocol handle that was invoked.
0x8000_0007 - 0x8000_0FFF		Reserved for future use.
0x8000_1000 - 0xFFFF_FFFF		Defined by the protocol that was invoked.

6 Core Protocol

All SVSM implementations must support the core protocol, which has the protocol ID value zero. The core protocol is versioned, permitting its extension over time; the initial version of the protocol is version 1. Versioning of the core protocol is strictly additive, i.e., all calls present in version 1 must be supported by all future implementations. The following table enumerates the set of calls supported by the core protocol:

Table 4: Core Protocol Services

Call ID	First version supported	Name	Function
0	1	SVSM_CORE_REMAP_CA	Remap the SVSM Calling Area to a new gPA.
1	1	SVSM_CORE_PVALIDATE	Execute PVALIDATE.
2	1	SVSM_CORE_CREATE_VCPU	Create a new vCPU.
3	1	SVSM_CORE_DELETE_VCPU	Delete a vCPU.
4	1	SVSM_CORE_DEPOSIT_MEM	Deposit additional memory for use by the SVSM.
5	1	SVSM_CORE_WITHDRAW_MEM	Withdraw unused memory no longer required by the SVSM.
6	1	SVSM_CORE_QUERY_PROTOCOL	Query the availability of a certain protocol.
7	1	SVSM_CORE_CONFIGURE_VTOM	Reconfigures the use of vTOM.

6.1 SVSM_CORE_REMAP_CA Call

This call is used to request that a new gPA be used for all future communication with the SVSM. It affects the Calling Area for calling vCPU only. The register RCX contains the requested gPA of the new Calling Area (8-byte value), which must be aligned to an 8-byte boundary.

Upon completion of the call, the SVSM_CALL_PENDING field of the previously configured Calling Area is cleared to indicate that the call has completed. In addition, if the call is successful, the SVSM will set the SVSM_CALL_PENDING field of the new Calling Area to zero so that a spurious invocation by an uncooperative host cannot trick the SVSM into thinking that another call was requested by the guest. The guest can examine RAX to determine whether the call was successful. If so, the previously configured Calling Area will no longer be examined by the SVSM and can be reused by the guest for any purpose.

6.2 SVSM_CORE_PVALIDATE Call

This call is used to request that the SVSM execute PVALIDATE on behalf of the guest. The register RCX contains the gPA of a page that holds a list of requested operations according to the format in the following table. This gPA must be aligned to an 8-byte boundary (or else the call will fail with SVSM_ERR_INVALID_PARAMETER).

Table 5: PVALIDATE Operation

Byte Offset	Size	Meaning
0x000	2 bytes	Number of entries in the list.
0x002	2 bytes	Index of the next entry in the list to be processed.
0x004	4 bytes	Reserved.
0x008	8 bytes	First entry in the list. Each entry specifies bits as follows: Bits 1:0 Value of RCX for the PVALIDATE operation (0=4 KB page, 1=2 MB page). Bit 2 Value of RDX for the PVALIDATE operation (0=make invalid, 1=make valid). Bit 3 Ignore EFLAGS.CF warnings. Bits 11:4 Reserved. Bits 63:12 gPA page number. Note that bits [20:12] must be zero if the entry describes 2 MB.
0x010	8 bytes	Second entry in the list, if any. Additional list entries follow.

The number of entries in the list must not be so large that the parameter list crosses a 4 KB boundary. The number of entries must be at least 1. If number of entries is not within a valid range, the call will return SVSM_ERR_INVALID_PARAMETER.

Upon entry to a call, the index of the next entry to be processed must be strictly less than the number of entries in the list; otherwise, the call will return SVSM_ERR_INVALID_PARAMETER.

Upon completion of a call, the index of the next entry to be processed will indicate the number of entries in the list that have been successfully processed. If the call returns SVSM_ERR_INCOMPLETE, then the SVSM was unable to process the entire list in a single operation, and the guest should reload RAX with the correct calling code (RCX will remain unmodified during the call) and issue the call again; the SVSM will continue processing based on the index of the next entry to be processed. If the call returns any other error, the index of the next entry will indicate the index of the entry that failed processing. If the call succeeds, the index of the next entry will be equal to the number of entries in the list.

The SVSM is expected to check that the guest is not attempting to execute PVALIDATE on a gPA that is assigned to the SVSM itself. If the SVSM detects that the guest is attempting to execute PVALIDATE on an address that is assigned to the SVSM, the call will return SVSM_ERR_INVALID_ADDRESS.

If an entry sets bit 1=0 (requesting invalidation of the page), the SVSM will first execute RMPADJUST to revoke permission for all VMPLs other than VMPL0. This is necessary to ensure that subsequent attempt to validate a page will observe a consistent VMPL permission state regardless of whether the host executes RMPUPDATE at any point in time.

If invocation of a PVALIDATE instruction results in the instruction completing with EFLAGS.CF=1, and if the entry that provoked the EFLAGS.CF=1 warning did not set the appropriate bit, the call will fail with the error code 0x8000_1010.

If invocation of a PVALIDATE instruction (or RMPADJUST instruction) results in the instruction completing with EAX != 0, the call will fail with an error code in the range 0x8000_1000 through 0x8000_100F, where the value is equal to (0x8000_1000 + EAX). PVALIDATE is architecturally specified to return EAX error codes in the range 0x0000-0x000F; if PVALIDATE unexpectedly returns a value outside of that range (e.g., due to architectural expansion of the error code space in a future revision), the call will fail with the error code 0x8000_1011.

If invocation of PVALIDATE completes successfully, and if the entry sets bit 1=1 (requesting validation of the page), the SVSM will additionally execute RMPADJUST to grant full permission to the VMPL of the vCPU making the request, as well as all more privileged VMPLs (numerically lower or equal to the requesting VMPL).

6.3 SVSM_CORE_CREATE_VCPU Call

This call is used to request creation of a new vCPU context. Register RCX contains the gPA of the page that will be used as the VMSA page for the new vCPU (8-byte value). Register RDX contains the gPA of the page that will be used for SVSM communication by the new vCPU (8-byte value). Both gPA values must be aligned to a 4 KB boundary or else the call will fail with SVSM_ERR_INVALID_PARAMETER. Register R8 contains the APIC ID associated with the new vCPU context.

In response to this call, the SVSM will first check that both gPA values specified do not conflict with any existing usage (SVSM private pages or any Calling Area address). If a conflict is detected, the call return will SVSM_ERR_INVALID_ADDRESS.

If the gPA values are valid, the SVSM will check the VMPL value specified in the new VMSA (after first revoking access to every VMPL other than itself to ensure that the VMSA page is not tampered with in the process of vCPU construction). If the VMPL value specified in the VMSA page is zero, the call will fail with SVSM_ERR_INVALID_PARAMETER. The SVSM will also check that EFER.SVME is set to one in the VMSA page; if not, the call will fail with

SVSM_ERR_INVALID_PARAMETER. If the VMSA checks pass, the SVSM will execute RMPADJUST to turn the page into a VMSA page so it can be used immediately. The SVSM will cache the gPA of the Calling Area associated with that vCPU for use by future calls to the SVSM.

Once a page is established as a VMSA page, it is treated as privately owned by the SVSM for the purpose of detecting memory usage conflicts. Any call which specifies the gPA of a VMSA page as an input gPA will fail with SVSM_ERR_INVALID_ADDRESS. This is also true of the VMSA of the startup vCPU. (This VMSA is not required to be within the initial contiguous range of pages assigned to the SVSM since the guest is expected to know where its own VMSA is located.)

6.4 SVSM_CORE_DELETE_VCPU Call

This call is used to delete a vCPU that was previously configured. Register RCX contains the gPA of the VMSA that is being deleted.

In response to this call, the SVSM will verify that the specified gPA belongs to a known VMSA address; if not, the call will return SVSM_ERR_INVALID_PARAMETER. If the gPA is valid, the SVSM will set VMSA.EFER.SVME=0; if this fails (because the VMSA is currently executing), the call will return a status code of 0x80001003 (equivalent to the FAIL_INUSE error code). If the VMSA is not currently executing, the SVSM will execute RMPADJUST to convert the page from a VMSA page to a normal page and ensure that full access is available to the VMPL of the requesting vCPU and all more privileged VMPLs (numerically lower than or equal to the requesting VMPL).

Once deletion of the vCPU is complete, the previously configured VMSA and Calling Area will no longer be examined by the SVSM and can be reused by the guest for any purpose.

The VMSA associated with the startup vCPU can never be deleted. An attempt to delete this vCPU will cause the call to return with SVSM_ERR_INVALID_PARAMETER.

6.5 SVSM_CORE_DEPOSIT_MEM Call

This call can be made to grant additional memory for exclusive use by the SVSM in case it requires additional memory to perform its work. The guest can know when additional memory is required because the SVSM will return a status code in the range 0x4nnn_nnnn, indicating the number of additional 4 KB pages required.

To make this call, register RCX is loaded with the gPA of a page that contains a list of gPA addresses that can be used by the SVSM according to the following table. This gPA must be aligned to an 8-byte boundary (or else the call will fail with SVSM_ERR_INVALID_PARAMETER).

Table 6: Deposit Memory Operation

Byte Offset	Size	Meaning
0x000	2 bytes	Number of entries in the list.
0x002	2 bytes	Index of the next entry in the list to be processed.
0x004	4 bytes	Reserved.
0x008	8 bytes	First entry in the list. Each entry specifies bits as follows: Bits 1:0 Size of the memory range described by this entry (0=4 KB page, 1=2 MB page). Bits 11:2 Reserved. Bits 63:12 gPA page number. Note that bits [20:12] must be zero if the entry describes 2 MB.
0x010	8 bytes	Second entry in the list, if any. Additional list entries follow.

The number of entries in the list must not be so large that the parameter list crosses a 4 KB boundary. The number of entries must be at least 1. If number of entries is not within a valid range, the call will return SVSM_ERR_INVALID_PARAMETER.

Upon entry to a call, the index of the next entry to be processed must be strictly less than the number of entries in the list; otherwise, the call will return SVSM_ERR_INVALID_PARAMETER.

Upon completion of a call, the index of the next entry to be processed will indicate the number of entries in the list that have been successfully processed. If the call returns SVSM_ERR_INCOMPLETE, then the SVSM was unable to process the entire list in a single operation, and the guest should reload RAX with the correct calling code (RCX will remain unmodified during the call) and issue the call again; the SVSM will continue processing based on the index of the next entry to be processed. If the call returns any other error, the index of the next entry will indicate the index of the entry that failed processing. If the call succeeds, the index of the next entry will be equal to the number of entries in the list.

For each entry in the list, the SVSM will verify that the memory described is not already private to the SVSM and does not overlap any page that has been configured as a Calling Area. If any overlap is detected, the call will return SVSM_ERR_INVALID_ADDRESS.

For each valid entry in the list, the SVSM will execute RMPADJUST to restrict VMPL permissions so that the pages are only accessible to VMPL0, making the pages private to the SVSM.

The call may return failure with the value of the next entry index not being zero. This indicates that some memory was successfully deposited with the SVSM, and some was not.

6.6 SVSM_CORE_WITHDRAW_MEM Call

This call permits the guest to reclaim memory that was made private to the SVSM but is no longer needed by the SVSM. Any SVSM operation that results in free memory that can be reclaimed will set the SVSM_MEM_AVAILABLE flag in the Calling Area of the startup vCPU.

To make this call, register RCX is loaded with the gPA of a page that can receive a list of memory that is no longer needed by the SVSM. This gPA must be aligned to an 8-byte boundary (or else the call will fail with SVSM_ERR_INVALID_PARAMETER). The page contents are ignored upon entry and populated during a successful call according to the following table.

Table 7: Withdraw Memory Operation

Byte Offset	Size	Meaning
0x000	2 bytes	Number of entries in the list.
0x002	6 bytes	Unused.
0x008	8 bytes x number of entries	List of gPA values of 4 KB pages that are no longer in use.

The maximum number of entries returned is constrained so that the returned list will not cross a 4 KB boundary. If the parameter page is aligned such that there is no room for any entries (i.e., the parameter page gPA is aligned to a 4 KB boundary plus 0xFF8 bytes), the call will return SVSM_ERR_INVALID_PARAMETER.

If no memory is available to withdraw, the number of entries will be zero. Unused entries beyond the end of the list are not zeroed.

When memory is withdrawn from the SVSM, the SVSM will first execute RMPADJUST to grant full access to the VMPL of the requesting vCPU and to all more privileged VMPLs (numerically lower than or equal to the requesting VMPL). Upon completion of the call, the SVSM will no longer access the pages, which can then be reused by the guest for any purpose.

Upon completion of the call, the SVSM_MEM_AVAILABLE flag of the Calling Area of the startup vCPU will be updated to indicate whether additional memory remains available for withdrawal.

This call will never return SVSM_ERR_INCOMPLETE. If the SVSM is unable to withdraw all available memory, the call will complete with SVSM_SUCCESS, and the SVSM_MEM_AVAILABLE flag of the startup vCPU's Calling Area will indicate that additional memory remains available for withdrawal.

6.7 SVSM_CORE_QUERY_PROTOCOL Call

This call is used to determine the availability of a given protocol. Bits [63:32] of register RCX contain the requested protocol number. Bits [31:0] of register RCX contain the desired version of the requested protocol. Upon completion of the call, register RCX is set to indicate availability of the requested protocol. If the protocol is unavailable at the requested version, register RCX will contain the value zero. If the protocol is available at the requested version, bits [63:32] of register RCX will support the maximum supported protocol version number, and bits [31:0] of register RCX will support the minimum supported protocol version number.

This call will always return SVSM_SUCCESS since the availability of the protocol is advertised through RCX. Querying for the presence of a protocol is not permitted to demand additional SVSM memory. (Calls to that protocol may request memory to be deposited.)

6.8 SVSM_CORE_CONFIGURE_VTOM Call

This call is used to query or reconfigure the use of vTOM on the calling processor. To support a transition between vTOM-based confidentiality and confidentiality determinations that rely exclusively on the Page Table Entry's C-bit, this call will also change the guest value of CR3, as well as RIP and RSP, to permit a clean transition from one environment to another.

SVSM_CORE_CONFIGURE_VTOM calls take two forms: query and configure, as indicated by bit zero of RCX. (RCX[0]=1 indicates query, while RCX[0]=0 indicates configure.) The register convention for the calls is described in the following table.

Table 8: vTOM Configuration Operation

Register	Contents		
RCX on entry	Query	Bit 0	Must be one.
		Bit 63:1	Must be zero.
	Configure	Bit 0	Must be zero.
		Bit 1	Set to zero to disable vTOM, or set to one to enable vTOM.
		Bit 2	If set to one, will cause VMSA.CR3 to be set to the value in RDX upon successful completion of the call.
		Bit 3	If set to one, will cause VMSA.RIP to be set to the value in R8 upon successful completion of the call.
		Bit 4	If set to one, will cause VMSA.RSP to be set to the value in R9 upon successful completion of the call.
		Bits 11:5	Must be zero.
		Bits 63:12	Bits 63:12 of the desired vTOM value. Must be zero if vTOM is being disabled.

Register	Contents	
RCX result	Bit 0	Will be zero.
	Bit 1	Will be one if vTOM configuration is supported; otherwise zero.
	Bits 11:2	Will be zero.
	Bits 19:12	vTOM alignment requirement as a power of two (value of 20 would indicate that vTOM must be aligned to a 1 MB boundary).
	Bits 63:20	Zero
RDX result	Minimum valid vTOM value if vTOM configuration is supported; otherwise undefined.	
R8 result	Maximum valid vTOM value if vTOM configuration is supported; otherwise undefined.	

If the call is successful, vTOM will be reported or reconfigured as requested. If vTOM is being reconfigured, then CR3 and RSP will be updated as requested, and execution will continue at the specified RIP with RAX containing the value SVSM_SUCCESS.

If the call is unsuccessful, no VMSA changes will occur, and execution will continue at the instruction following the VMGEXIT with RAX containing the appropriate error code.

The call may fail if any reserved bit RCX is set inappropriately; this will result in the call returning SVSM_ERR_INVALID_PARAMETER.

The SVSM may choose to deny the call if it cannot support the request. For example, the SVSM may be unable to reconfigure VTOM if more than a single vCPU has been configured or if the requested configuration differs from configurations present on other vCPUs. This will result in the call returning SVSM_ERR_INVALID_REQUEST. If the value of vTOM is one that cannot be supported by the hosting environment, then the call will result in SVSM_ERR_INVALID_ADDRESS.